Subject: Re: bug in CoWork since C++11
Posted by crydev on Sat, 02 Jul 2016 15:49:05 GMT
View Forum Message <> Reply to Message

I have changed my synchronization method to something more safe, leaving the workers not doing any synchronization work at all. I changed it because I thought my synchronization was faulty. However, it does not appear to be faulty, because the bug remains, even in build 9994.

I start my workers like this, where this->mWorkerFileOrder is a Vector containing 8 structures that describe the work the worker should do.

```
// Launch the workers.
for (auto& work : this->mWorkerFileOrder)
{
 threadPool & THISBACK2(FirstScanWorker<T>, &work,
((T)(reinterpret_cast<ScanParameters<T>*>(GlobalScanParameter))->ScanValue));
}
```

The FirstScanWorker function looks like this; the function does not write to memory that does not belong to the worker it represents. Every worker has a thread-local variable called FinishedWork, which is set to true when the worker completes. Another program component peeks this variable for every worker, and finalizes when all workers are done.

```
// Represents the default template worker function for the set of workers including specialized ones.
// This set of workers run the first scan sequence.
template <class T>
void MemoryScanner::FirstScanWorker(WorkerRegionParameterData* const regionData, const T& value)
{
#ifdef _DEBUG
 LARGE_INTEGER frequency;
 LARGE_INTEGER t1;
 LARGE_INTEGER t2;

 // Get the amount of ticks per second.
 QueryPerformanceFrequency(&frequency);

 // Start the timer.
 QueryPerformanceCounter(&t1);
#endif

 // ------------------------------------------------

 FileOut addressesFile(AppendFileName(mMemoryScanner->GetTempFolderPath(),
```

```
Format("Addresses%i.temp", regionData->WorkerIdentifier)));
 FileOut valFile(AppendFileName(mMemoryScanner->GetTempFolderPath(),
Format("Values%i.temp", regionData->WorkerIdentifier)));

 int fastScanAlignSize = GlobalScanParameter->CurrentScanFastScan ? sizeof(T) : 1;
 if (fastScanAlignSize == sizeof(__int64))
 {
  fastScanAlignSize = sizeof(int);
 }

 unsigned int fileIndex = 0;
 const unsigned int forLoopLength = regionData->OriginalStartIndex + regionData->Length;

 for (unsigned int i = regionData->OriginalStartIndex; i < forLoopLength; ++i)
 {
  unsigned int arrayIndex = 0;
  unsigned int currentArrayLength = 256;

  MemoryRegion& currentRegion = this->memRegions[i];
  const SIZE_T regionSize = currentRegion.MemorySize;
  currentRegion.FileDataIndexes.StartIndex = fileIndex;

  SIZE_T* localAddresses = NULL;
  T* localValues = NULL;

  Byte* buffer = new Byte[currentRegion.MemorySize];
  if (CrySearchRoutines.CryReadMemoryRoutine(this->mOpenedProcessHandle,
(void*)currentRegion.BaseAddress, buffer, currentRegion.MemorySize, NULL))
  {
   for (SIZE_T i = 0; i < regionSize; i += fastScanAlignSize)
   {
    const T* tempStore = (T*)&(buffer[i]);

    if ((*reinterpret_cast<ValueComparator<T>*>(this->mCompareValues))(*tempStore, value))
    {
     if (!localAddresses || !localValues)
     {
      localAddresses = new SIZE_T[currentArrayLength];
      localValues = new T[currentArrayLength];
     }

     if (arrayIndex >= currentArrayLength)
     {
      const unsigned int oldCurrentArrayLength = currentArrayLength;
      this->ReallocateMemoryScannerBufferCounter(&currentArrayLength);

      SIZE_T* newAddressesArray = new SIZE_T[currentArrayLength];
      memcpy(newAddressesArray, localAddresses, oldCurrentArrayLength * sizeof(SIZE_T));
```

```
     delete[] localAddresses;
     localAddresses = newAddressesArray;

     T* newValuesArray = new T[currentArrayLength];
     memcpy(newValuesArray, localValues, oldCurrentArrayLength * sizeof(T));
     delete[] localValues;
     localValues = newValuesArray;
    }

    localAddresses[arrayIndex] = currentRegion.BaseAddress + i;
    localValues[arrayIndex++] = *tempStore;

    ++fileIndex;
   }
  }
 }

 delete[] buffer;

 if (arrayIndex > 0)
 {
  this->mScanResultCount += arrayIndex;

  if (CachedAddresses.GetCount() < MEMORYSCANNER_CACHE_LIMIT)
  {
   AddResultsToCache(arrayIndex, localAddresses, NULL);
  }

  addressesFile.Put(localAddresses, arrayIndex * sizeof(SIZE_T));
  delete[] localAddresses;

  valFile.Put(localValues, arrayIndex * sizeof(T));
  delete[] localValues;
 }
 else
 {
  if (localAddresses)
  {
   delete[] localAddresses;
   delete[] localValues;
  }
 }

 currentRegion.FileDataIndexes.ResultCount = arrayIndex;
 this->UpdateScanningProgress(AtomicInc(RegionFinishCount));
}

addressesFile.Close();
```

```
 valFile.Close();

 // Indicate that this worker is done processing.
 regionData->FinishedWork = true;

 // ------------------------------------------------

#ifdef _DEBUG
 // Stop the timer.
 QueryPerformanceCounter(&t2);
 OutputDebugString(Format("Worker %i took %f ms\r\n", regionData->WorkerIdentifier,
(t2.QuadPart - t1.QuadPart) * 1000.0 / frequency.QuadPart));
#endif
}
```

The bug still remains; most of the time, not all workers finish, because CoWork thinks it doesn't have anything to do anymore. Sometimes it finishes, but rarely. :)

Thanks, and my apologies for the late response. I haven't had time. :(

crydev
